2015 12th International Conference on Information Technology - New Generations

Visual approach for change impact analysis: a controlled experiment

Fernanda M. Delfim
Dept. de Ciência da Computação
Universidade Federal de Uberlândia
Uberlândia, Brazil
fer.madeiral@gmail.com

Lilian P. Scatalon
Dept. de Sistemas de Computação
Universidade de São Paulo
São Carlos, Brazil
lilian.scatalon@usp.br

Jorge M. Prates and Rogério E. Garcia
Dept. de Matemática e Computação
Universidade Estadual Paulista
Presidente Prudente, Brazil
jorgemprates@gmail.com; rogerio@fct.unesp.br

***Abstract*—In the context of Software Maintenance, when a source code element must be changed, there is the need to identify if other elements will be affected by the change, in order to keep the code consistent. This identification is performed during the activity of change impact analysis. Aiming to support maintainers during this activity, software visualization tools allow a visual exploration of source code elements. In this paper, we present a study aimed at evaluating the support provided to change impact analysis by visual representations of Java program elements and their associations. To this end, we conducted a controlled experiment involving 24 undergraduate students, comparing the visual support approach and an ad hoc approach, where only the source code is analyzed to estimate impact change. Results showed that the effectiveness obtained by using the visual approach is significantly superior. This is an indication that visual support should be considered to change impact analysis aiming at reducing software maintenance costs.**

Keywords*—*Software Maintenance; Program Comprehension; Change Impact Analysis; Software Visualization; Controlled Experiment.

I. Introduction

In order to accomplish a change during Software Maintenance, it is necessary to ensure that new defects are not introduced and the code remain consistent [1]. The consistence across elements impacted is supported by the *change impact analysis* activity [2], one of software maintenance steps.

Analyzing the impact of changing a software system element (instruction, method, class, package) involves the identification of other elements related to it. This can be done systematically, applying a technique composed by rules that define which are the dependencies for each change type [3].

In this sense, a change impact analysis technique provides guidance about what are the dependencies to look for (for example, a rule can state that if you change a method, its caller methods can be potentially affected), but the maintainer still needs to identify them throughout the source code. An ad hoc approach to do that uses an IDE (*Integrated Development Environment*) as support, looking at the raw code searching for the potentially impacted elements.

On the other hand, the maintainer can have benefits by using visualization software techniques (and tools) – visualization can emphasize dependencies among software elements [4]. The ability to visualize all the code elements and their dependencies can help a maintainer to perform change impact analysis [4].

In this study, we investigate through a controlled experiment if visualizing dependencies indeed help the maintainer. More specifically, we investigate the effect of visual support on effectiveness achieved by the maintainer, while performing change impact analysis. The ad hoc approach (using Eclipse IDE and its resources, looking at raw code) is compared to the visual approach (using $SoftVis_{4CA}$, a software visualization tool developed by [5]).

The remainder of this paper reports the controlled experiment and its results. In Section II, we briefly explain the change impact analysis technique and the visual approach used in this study. Sections III and IV provide details about experiment planning and execution, respectively. Section V presents the results, which are discussed with respect to validity in Section VI. Finally, Section VIII presents conclusions and further work.

II. Background

A. Change impact analysis

The change impact analysis process is composed by three main steps [6]: 1) analysis of the change request and the software; 2) estimation of the potential change effects; and 3) implementation of the change request. In the first step, the maintainer analyzes the software and identify a set of code elements that could be initially affected by implementation of the change request, named *change set*.

Next, in the second step, the maintainer estimates what are the other elements potentially affected by the elements in the *change set*, composing the *estimated impact set*. Finally, in the third step, the maintainer implements the change request, considering both *change set* and *estimated impact set*. Elements are actually modified only if there is indeed the need to change, what characterizes the *actual impact set*.

In particular, the second step is performed by means of some change impact analysis technique. Moreover, this second step is the focus of this study. There are many different change impact analysis techniques in the literature [3]. For each technique, different software artifacts (e.g. requirements, design, source code, test cases) and software elements (e.g. files, classes, attributes, methods, instructions) are considered. In our study, subjects applied Sun's et al. technique [1], which

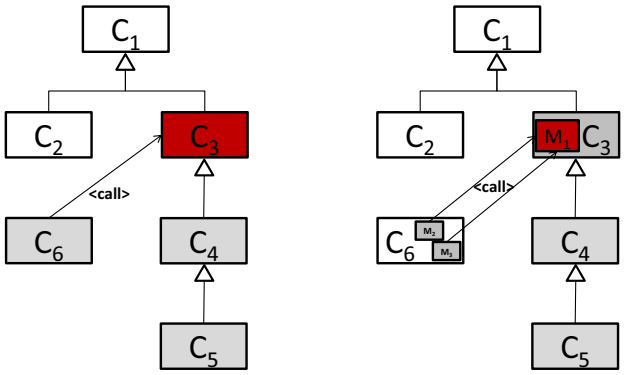
978-1-4799-8828-0/15 $31.00 © 2015 IEEE
DOI 10.1109/ITNG.2015.69

limits the scope of the analysis for source code level and defines rules to estimate impact for elements of object-oriented programs.

This technique is composed by a set of impact rules for each change type in code (e.g. change class name, delete a class, etc). Two change types defined by Sun et al. [1] were used in the experiment: *Change name of the class* (CNC) and *Change name of the method* (CNM). The corresponding impact rules are illustrated in Fig. 1, with classes and methods linked by dependencies.

Concerning CNC (Fig. 1(a)), let C_3 be the class whose name must be changed. The impact set is formed by (i) classes that directly inherit C_3 (C_4), (ii) classes that call methods of C_3 (C_6) and (iii) classes that indirectly inherit C_3 (C_5). Hence, the final impact set is $\{C_4, C_5, C_6\}$.

In a similar way, for CNM (Fig. 1(b)), let M_1 be the method whose name must be changed. The impact set is formed by (i) the owner class of M_1 (C_3), (ii) methods that call M_1 (M_2 and M_3) and (iii) classes that directly and indirectly inherit the owner class of M_1 (C_4 and C_5). Then, the final impact set is $\{C_3, M_2, M_3, C_4, C_5\}$.



(a) Change name of the class (b) Change name of the method

Figure 1. Change impact

B. Change impact analysis support

According to Li et al. [3], several tools have been developed aiming to support the application of change impact analysis techniques, such as JRipples [7]. JRipples is an Eclipse plug-in to assist in static analysis for incremental changes. Given an element to be changed (for example, a class), JRipples provides as output the list of impacted elements.

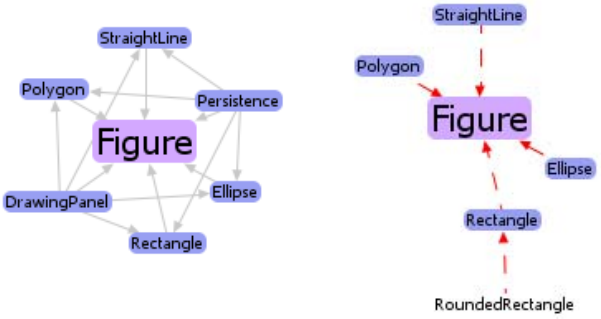
However, the tool does not emphasize the dependencies among elements. In other words, it is not clear *how* the output elements are related to the element to be changed. Similarly, other tools also put on the user the onus to figure out the dependencies underlying output elements.

Aiming to address this problem, software visualization can be useful supporting program comprehension, taking advantage of human ability to deal with visual information [8]. Hence, a visual exploration of program elements and their dependencies can be promising in change impact analysis.

In this scenario, in [5] we proposed a visual approach to aid comprehension of Java programs. The idea is to represent program elements (packages, classes and methods) and their dependencies in interactive visual representations (or views).

This visual approach has been implemented in $SoftVis_{4CA}$, a tool that provides six views of program elements at different levels of details. Moreover, the views are coordinated, i.e., if some element is selected in a view, the corresponding or related elements in other views are highlighted.

$SoftVis_{4CA}$ allows visualizing dependencies among elements, such as classes and methods (the ones used in the experiment). Fig. 2 shows the *Inter-Classes View*, where nodes represent classes and edges represent calls between methods of classes (Fig. 2(a)) or inheritance (Fig. 2(b)). Thus, the *Inter-Classes View* provides necessary information to analyze the impact of CNC change type and to understand how the elements are related.



(a) Call edges (b) Inheritance edges

Figure 2. *Inter-Classes View*

Similarly, the *Inter-Methods View* in Fig. 3 presents calls between methods – the larger node is the one that represent a method in focus (in inspection), the blue nodes represent its caller methods and the green nodes represent its called methods. Since the views are coordinated, *Inter-Methods View* and *Inter-Classes View* provides necessary information to analyze and understand the impact of CNM change type.

Figure 3. *Inter-Methods View*

C. Related work

The main idea of this paper is to combine software visualization and change impact analysis. This idea can be identified in Hutchins and Gallagher [9] study, that proposed the visual representation of program slices and their dependencies for change impact analysis in C programs.

It is worth mentioning that impact analysis in the instruction level (using program slices) is recurring in the literature [9]–[11] and, generally, focusing on C or C++ programming languages. Experiments evaluating techniques that use program slicing also can be found in the literature [12]–[15].

Nevertheless, the use of visualization techniques has been intensified in the context of Software Evolution, where changes already performed through several versions of a program are emphasized with visual resources [8], [16]–[20].

III. Experiment planning

A. Experimental context

The experiment was executed at São Paulo State University (Universidade Estadual Paulista – UNESP), campus of Presidente Prudente. Undergraduates in Computer Science composed our subject sampling (n=24). The students were volunteers and were not evaluated by their performance. Most of them were familiarized with Eclipse (19 subjects) and never used software visualization techniques before (18 subjects).

As experiment objects, we used Java code (in Eclipse) and the corresponding bytecode (in $SoftVis_{4CA}$) of the following systems: (i) HealthWatcherOO, a system that allows citizens to register complaints about health institutions [21]; and (ii) JUnit[1] (version 4), a framework for writing and running Java test cases. Both of them are real systems and, therefore, provide a reasonable structure (see Table I) to be evaluated in order to perform a change.

TABLE I. Metrics of experiment objects

System	# classes	# methods	# LOC
HealthWatcherOO	102	746	5560
JUnit	232	1315	4965

B. Hypotheses

The main goal of our study is to analyze if a visual approach is more effective than an ad hoc approach during change impact analysis. Therefore, the hypotheses (see Table II) were formulated in order to verify the effect of the adopted approach (ad hoc or visual) in effectiveness (subject ability to find the estimated impact set for some required change).

TABLE II. Investigated Hypotheses

Hypothesis type	Formalized hypothesis
Null hypothesis	H_0: *Effectiveness(visual)* = *Effectiveness(ad hoc)*
Alternative hypothesis	H_a: *Effectiveness(visual)* > *Effectiveness(ad hoc)*

C. Variables

We investigated one *factor* (independent variable): the *approach* used while determining estimated impact set. This factor has two treatments: ad hoc approach (using Eclipse) and visual approach (using $SoftVis_{4CA}$).

The outcome observed (dependent variable) in the experiment was the *effectiveness*, which, in this case, is related to the number of correctly identified elements in the estimated impact set, according to rules defined by Sun et al. [1].

By comparing answers provided and correct answers (as an oracle), we evaluated subjects' performance using the metrics *precision* and *recall* [22]. All answers from each subject were aggregated using (1) and (2) [23] as follows:

$$precision_p = \frac{\sum_i |answered_{p,i} \cap correct_i|}{\sum_i |answered_{p,i}|} \quad (1)$$

$$recall_p = \frac{\sum_i |answered_{p,i} \cap correct_i|}{\sum_i |correct_i|} \quad (2)$$

where $answered_{p,i}$ is the set of elements provided as answer by subject p to an element i required to change, and $correct_i$ is the corresponding set of correct elements.

Since both $precision$ and $recall$ must represent effectiveness, the metric $F\text{-}measure$ [22] was applied (3). Thus, the $F\text{-}measure$ value is the $Effectiveness$ (4).

$$F\text{-}measure_p = 2 * \frac{precision_p * recall_p}{precision_p + recall_p} \quad (3)$$

$$Effectiveness_p = F\text{-}measure_p \quad (4)$$

All these metrics ($precision$, $recall$, $F\text{-}measure$ and, consequently, $Effectiveness$) assume values in the interval [0,1].

D. Experimental design

Since the experiment had one factor with two treatments, a predefined design fits: the paired comparison design [24]. In this kind of design, each subject applies both treatments on the same object. However, this can cause learning effects: analyzing the same code, even with a different approach on the second time, can provide different results because the subject will be then familiarized with the code. So, this is the reason why we used two experiment objects.

The experimental design is presented in Table III: subjects were distributed in two groups (groups A and B, with 12 subjects in each) and each one applied both ad hoc and visual approach on different systems.

TABLE III. Experimental design

System	Ad Hoc Approach	Visual Approach
HealthWatcherOO	Group A	Group B
JUnit	Group B	Group A

[1] http://junit.org

IV. Experiment Execution

The execution was performed in two sessions over two days, in order to diminish subject fatigue. Table IV shows the adopted procedures and respective time spent. Basically, the first session comprised the experiment preparation and subject training. In the second session, the experiment tasks were applied and we collected results.

Experiment tasks aimed to exercise the change impact analysis activity. The subjects received a list of program elements to be modified with the specification of what change should be performed (we considered only CNC and CNM change types – see Section II-A). For each one, they should provide the corresponding set of program elements potentially affected, applying some of the support approaches with the technique developed by Sun et al. [1].

TABLE IV. Execution schedule

Day 1 - Preparation		
5 min	Experiment presentation	
5 min	Distribution of consent forms	
10 min	Distribution of profile questionnaires	
30 min	Training in change impact analysis	
30 min	Training in $SoftVis_{4CA}$ tool	
Day 2 - Tasks		
10 min	Subject groups organization	
	Ad hoc approach	Visual approach
30 min	Group A	Group B
30 min	Group B	Group A

V. Data Analysis

From the collected data, we calculated $Effectiveness$ for each subject, as defined in Section III-C. Table V summarizes results in descriptive statistics. Only from these values, it is possible to observe that the effectiveness of subjects was higher using the visual approach.

TABLE V. Descriptive analysis for $Effectiveness_p$

Approach	mean	median	standard deviation
Ad hoc	0.67559	0.66667	0.22463
Visual	0.8064	0.875	0.22766

The $Effectiveness$ distribution of both approaches can be observed in the boxplot shown in Fig. 4. Analyzing the distances of the median from the upper and lower quartiles of the ad hoc approach, it is possible to see that there is a bias at the top of the box, meaning that the $Effectiveness$ values are not symmetric in the middle of the distribution. Instead, the distribution in the middle is positively skewed, meaning that the $Effectiveness$ values are more concentrated at the bottom ([0.59, 0.67]) than at the top. On the other hand, the box of the visual approach present a negative skewness very small, meaning that the distribution of the data in the middle are very close to be symmetric. Moreover, the interval the concentration of values in visual approach is [0.88, 1], meaning that the region of concentration in visual approach is of highest values than the concentration in ad hoc approach.

Observing the whiskers drawn outside the box, it is possible to observe the data skewness in the tails to bottom. For the two approaches, the data distribution is negatively skewed, meaning that there is a greater range data on the top. In visual approach box, there is an extreme case of skewness: the upper quartile is equal to the maximum value – there is a high concentration of high values in the distribution of $Effectiveness$ of the participants. Also, it is possible to observe that two subjects using visual approach were identified as outliers (small values).

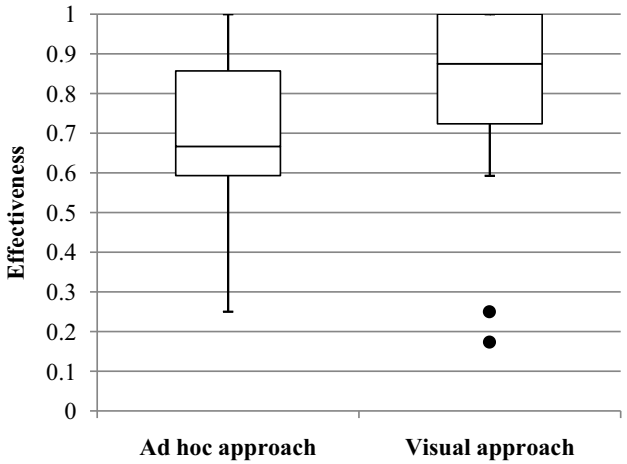


Figure 4. $Effectiveness_p$ values distribution in a boxplot

In Fig. 5, the results are displayed as a scatter plot, allowing to indicate individually $Effectiveness_p$. Each projected point represents a subject p, and the x and y coordinates are values of $Effectiveness_p$ using ad hoc and visual approach, respectively. One can notice that, in general, subjects performed well with both approaches. There is a concentration of points above the value of 0.5 for both axes.

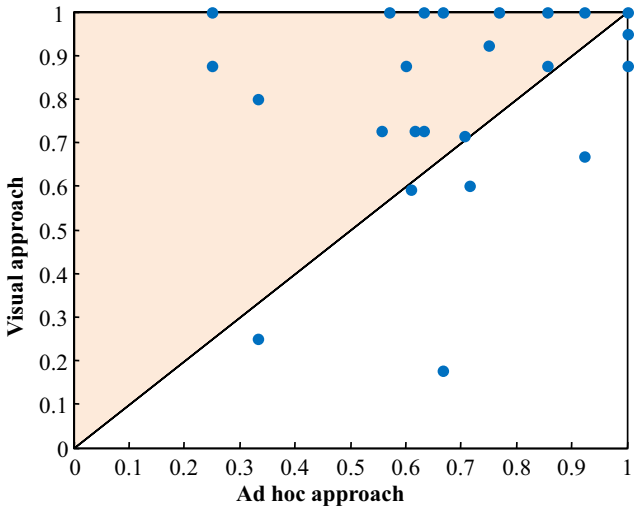


Figure 5. $Effectiveness_p$ values in a scatter plot

Additionally, the Wilcoxon test (*Wilcoxon Signed-Rank* [25]) was applied in order to evaluate the hypotheses (Section III-B). This test was chosen because the probability distribution of the variables is unknown and because the experimental design was paired, where each group used a different object in each approach. It is noteworthy that the two outliers shown in the boxplot were not excluded for the test calculation.

The sum of ranks for ad hoc approach (positive ranks)

was 74.5 and for visual approach (negative ranks) was 225.5. The critical value of the Wilcoxon test for 24 samples (in the context of the experiment, the samples are participants) is 81 for α = 0.05. Since the smallest sum of ranks is lesser than the critical value, the null hypothesis H_0 can be rejected and the approach with highest sum of ranks is considered superior. Therefore, in our study, the visual approach is more effective than the ad hoc approach, which indicates that the alternative hypothesis H_a can be accepted, at 95% confidence level.

VI. Threats to validity

A fundamental issue of the experiment is the validity of the results [24]. In this work, the threats to validity are discussed according to the following validity types [24]: internal, external, construction and conclusion. In parallel, we indicate choices made throughout the experimental process in order to deal with them.

The *internal validity* ensures that the results are influenced only by controlled and measured factors. A threat of this type would be the subject lack of ability to perform the proposed tasks, since they were not familiarized with the involved techniques. In this sense, we offered trainings in order to prepare them. Moreover, during the execution tasks, they had access to training material and experimenters were available in case they had doubts.

Another threat to internal validity would be ambiguous answers, what could result in incorrect processing of the collected data. Thus, we selected only elements with a unique identifier in code. Other threat can be the limited time to perform tasks. However, all subjects finished within the stipulated time.

The *external validity* refers to the ability to generalize the results. The main threat to validity of this study is related to the subject representativeness. They were students instead of practitioners (professionals). So, additional studies are required to mitigate this threat.

The *construct validity* ensures that the cause-effect relationship in theory is well represented in the experiment. The construction of the cause-effect relationship can be summarized as follows: ad hoc and visual approaches were evaluated, represented by Eclipse IDE and the $SoftVis_{4CA}$ tool, respectively, and the observed effect was the effectiveness of each participant.

In this scenario, one should distinguish between the tool we used ($SoftVis_{4CA}$) and a visual approach. There are other Software Visualization tools [26] aimed at supporting program comprehension, which can be applied in this situation. Nevertheless, the $SoftVis_{4CA}$ tool provides visualization of dependencies among program elements, which enhances change impact analysis support.

Still considering this threat, the choice of an IDE to be used as ad hoc approach is justified by the intent to evaluate the difference of having the support of a visual representation of the dependencies. In the ad hoc approach, the dependencies are not emphasized in any way, only the source code is displayed. It is noteworthy that during the tasks using visual approach, the participants did not have access to the objects source code. The $SoftVis_{4CA}$ tool generates visual representations from the bytecode of the programs.

Another threat related to construct validity is evaluating only the effectiveness, when other aspects of the approaches also should be evaluated in order to fully investigate the support approach used in change impact analysis. This threat can be mitigated with other complementary experiments on this topic. The specific goal of this study was to evaluate observed effectiveness.

Conclusion validity is related to the ability to obtain a correct conclusion from the experiment results. Some threats that can be pointed out are lack of reliability of the measures and problems with hypothesis testing, such as low statistical power and violated assumptions of the adopted test.

Regarding these threats, the chosen measures to apply in results are objective, i.e., they can be repeated to the same type of collected data. Also, no assumptions were violated. The chosen statistical test (Wilcoxon) does not require that the variables have a specific distribution – it is a non-parametric test. Furthermore, the reliability of the test result was increased because each group used a different object, what prevents familiarity with objects to favor any approach (learning effect).

VII. Lessons learned

Carrying out this study allowed us to gain some experiences, which are shared in this section. Regarding subject training, besides presenting a lecture about $SoftVis_{4CA}$ visual representations, we consider appropriate to assign practical exercises to subjects, in order to guarantee that they mastered the material and will be fully able to perform the proposed tasks.

From the two outliers in Fig. 4, for example, it was possible to infer that there was some misunderstanding about the visual representation of inheritance (directed edges on the graph). After the experiment they reported be confused about arrow direction. Practical exercises would allow us to identify difficulties like that.

Additionally, we spent a lot of effort distributing the experiment material to subjects and collecting back the answers at the end of the tasks. An alternative to handle this is using an experimentation support tool like the ExpTool [27].

VIII. Conclusions

The experiment presented in this paper is a preliminary study that investigated visual support to perform change impact analysis. From the experiment it is possible to claim that a visual approach (emphasizing dependencies among program elements) provides more effectiveness to the maintainer than an ad hoc approach (using only an IDE as resource) to analyze change impact. The visual approach is represented by $SoftVis_{4CA}$ tool, that provides visualization of the dependencies between code elements. The ad hoc approach is represented by the use of the Eclipse IDE.

Regarding related works, the study presents two important distinctions. The first one is the use of Software Visualization to allow analyzing the impact of a change *before* it is performed. In general, related studies investigate visualization to highlight changes *already made* over several versions of a program (Software Evolution). The second one is the investigation of the effectiveness provided by the approaches considering

human subjects. Other studies focus on the accuracy provided by the analytical techniques, with no subjects (human being) – analytical techniques are applied to systems in order to evaluate change impact analysis, using the effectiveness obtained.

Moreover, this study can fit in the more general context to investigate the increase in productivity and the reduction of software maintenance cost, since program comprehension mechanisms, such as software visualization, have influence on these aspects [28].

As future work, we intend to investigate the visual approach for change impact analysis considering other program elements. In particular, $SoftVis_{4CA}$ also generates a visual representation of program slices and their dependencies. This allows to investigate impact analysis at instruction level.

We also intend to evaluate other aspects, as the efficiency provided by different approaches – focusing on time, we intend to investigate how quickly the maintainer identify change impact with these approaches. Performing other studies addressing complementary aspects allows consolidating knowledge about the visual support for this activity in software maintenance.

ACKNOWLEDGMENT

We would like to thank Rafael Giusti for the help with the experiment instrumentation and data analysis. Additionally, we also thank the students involved in the experiment as subjects.

REFERENCES

[1] X. Sun, B. Li, C. Tao, W. Wen, and S. Zhang, "Change Impact Analysis Based on a Taxonomy of Change Types," in *34th Annual IEEE Computer Software and Applications Conference (COMPSAC)*, July 2010, pp. 373–382.

[2] B. G. Ryder and F. Tip, "Change Impact Analysis for Object-Oriented Programs," in *2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE '01)*. New York, NY, USA: ACM, 2001, pp. 46–53.

[3] B. Li, X. Sun, H. Leung, and S. Zhang, "A survey of code-based change impact analysis techniques," *Software Testing, Verification and Reliability*, vol. 23, pp. 613–646, December 2013.

[4] F. M. Delfim and R. E. Garcia, "Uma Proposta de Múltiplas Visões Coordenadas para Apoiar Análise de Impacto de Mudança," in *X Workshop de Manutenção de Software Moderna (WMSWM '13)*, 2013, pp. 9–16.

[5] F. M. Delfim and R. E. Garcia, "Multiple Coordinated Views to Support Aspect Mining Using Program Slicing," in *25th International Conference on Software Engineering and Knowledge Engineering (SEKE '13)*, 2013, pp. 531–536.

[6] S. A. Bohner, "Software Change Impacts – An Evolving Perspective," in *International Conference on Software Maintenance (ICSM '02)*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 263–272.

[7] J. Buckner, J. Buchta, M. Petrenko, and V. Rajlich, "JRipples: A Tool for Program Comprehension during Incremental Change," in *13th International Workshop on Program Comprehension (IWPC '05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 149–152.

[8] S. Diehl, *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[9] M. Hutchins and K. Gallagher, "Improving Visual Impact Analysis," in *1998 International Conference on Software Maintenance (ICSM)*, November 1998, pp. 294–303.

[10] P. Tonella, "Using a Concept Lattice of Decomposition Slices for Program Understanding and Impact Analysis," *IEEE Transactions on Software Engineering*, vol. 29, no. 6, pp. 495–509, June 2003.

[11] M. Acharya and B. Robinson, "Practical Change Impact Analysis Based on Static Program Slicing for Industrial Software Systems," in *33rd International Conference on Software Engineering (ICSE '11)*. New York, NY, USA: ACM, 2011, pp. 746–755.

[12] J. Law and G. Rothermel, "Whole Program Path-Based Dynamic Impact Analysis," in *25th International Conference on Software Engineering (ICSE '03)*. Washington, DC, USA: IEEE Computer Society, May 2003, pp. 308–318.

[13] A. Orso, T. Apiwattanapong, and M. J. Harrold, "Leveraging Field Data for Impact Analysis and Regression Testing," *ACM SIGSOFT Software Engineering Notes*, vol. 28, no. 5, pp. 128–137, September 2003.

[14] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley, "Chianti: A Tool for Change Impact Analysis of Java Programs," in *19th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA '04)*. New York, NY, USA: ACM, 2004, pp. 432–448.

[15] L. Badri, M. Badri, and D. St-Yves, "Supporting Predictive Change Impact Analysis: A Control Call Graph Based Technique," in *12th Asia-Pacific Software Engineering Conference (APSEC '05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 167–175.

[16] M. Lanza, "The Evolution Matrix: Recovering Software Evolution using Software Visualization Techniques," in *4th International Workshop on Principles of Software Evolution (IWPSE '01)*. New York, NY, USA: ACM, 2001, pp. 37–42.

[17] C. Collberg, S. Kobourov, J. Nagra, J. Pitts, and K. Wampler, "A System for Graph-Based Visualization of the Evolution of Software," in *2003 ACM Symposium on Software Visualization (SoftVis '03)*. New York, NY, USA: ACM, 2003, pp. 77–86.

[18] F. Van Rysselberghe and S. Demeyer, "Studying Software Evolution Information by Visualizing the Change History," in *20th IEEE International Conference on Software Maintenance (ICSM '04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 328–337.

[19] H. C. Gall and M. Lanza, "Software Evolution: Analysis and Visualization," in *28th International Conference on Software Engineering (ICSE '06)*. New York, NY, USA: ACM, 2006, pp. 1055–1056.

[20] R. L. Novais, G. de F. Carneiro, P. R. M. Simões Júnior, and M. G. Mendonça, "On the Use of Software Visualization to Analyze Software Evolution: An Interactive Differential Approach," in *Enterprise Information Systems*, ser. Lecture Notes in Business Information Processing, R. Zhang, J. Zhang, Z. Zhang, J. Filipe, and J. Cordeiro, Eds. Springer Berlin Heidelberg, 2012, vol. 102, pp. 241–255.

[21] A. Costa Neto, M. Ribeiro, M. Dósea, R. Bonifácio, P. Borba, and S. Soares, "Semantic Dependencies and Modularity of Aspect-Oriented Software," in *First International Workshop on Assessment of Contemporary Modularization Techniques (ACoM '07)*. Washington, DC, USA: IEEE Computer Society, May 2007, pp. 11–16.

[22] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[23] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, "Mining Version Histories to Guide Software Changes," in *26th International Conference on Software Engineering (ICSE '04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 563–572.

[24] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction*. Boston, USA: Kluwer Academic Publishers, 1999.

[25] J. Demšar, "Statistical Comparisons of Classifiers over Multiple Data Sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[26] P. Caserta and O. Zendra, "Visualization of the Static Aspects of Software: A Survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 7, pp. 913–933, July 2011.

[27] J. Pucci Neto, L. P. Scatalon, R. E. Garcia, R. C. M. Correia, and C. Olivete Junior, "ExpTool: A Tool to Conduct, Package and Replicate Controlled Experiments in Software Engineering," in *12th International Conference on Software Engineering Research and Practice*, 2014, pp. 1–6.

[28] J. R. Cordy, "Comprehending Reality – Practical Barriers to Industrial Adoption of Software Maintenance Automation," in *11th IEEE International Workshop on Program Comprehension (IWPC '03)*. Washington, DC, USA: IEEE Computer Society, May 2003, pp. 196–205.